Méthodes de Monte Carlo avec réduction de variance

Application aux Options Basket

Théo Fromentin

Octobre 2024

Table des matières

1 Contexte

On considère un modèle de Black-Scholes multidimensionnel pour l'évolution de $d \geq 1$ actifs financiers $(S^1, \ldots, S^d)$ sur $[0, T]$. Ainsi pour $1 \leq i \leq d$ l'actif $(S_t^i)_{t\in[0,T]}$ suit l'EDS suivante :

$$\mathrm{d}S_t^i = rS_t^i\mathrm{d}t + \sigma_i S_t^i \mathrm{d}W_t^i, \quad S_0^i = s_0^i \in \mathbb{R} \tag{1}$$

où $r > 0$ représente le taux d'intérêt commun à tous les actifs et σ_i représente la volatilité propre du i-ème actif. Le processus $(W_t)_{t\in[0,T]}$ est un mouvement brownien multidimensionnel sur $\mathbb{R}^d$ centré de matrice de corrélation $\rho \in [-1, 1]^{2d}$ telle que

$$\mathrm{d}\langle W^i, W^j\rangle_t = \rho_{ij}\mathrm{d}t, \quad \rho_{ij} \in [-1, 1].$$

Le but est de calculer le prix d'un call sur indice de prix I_t défini par :

$$\forall t \in [0, T], \quad I_t = \sum_{i=1}^{d} a_i S_t^i,$$

avec $a_1 + \cdots + a_d = 1$. On va donc mettre en oeuvre une méthode de Monte Carlo pour calculer

$$\mathbb{E}[e^{-rT}(I_T - K)^+]$$

2 Le modèle de classe `PriceIndex`

Dans tout le reste du projet j'ai fait le choix d'utiliser la bibliothèque d'algèbre linéaire `Eigen`. D'autre choix sont possibles, comme `Armadillo` par exemple.

Nous allons implémenter un modèle de classe au sein du fichier `price_index.hpp`, ainsi de pouvoir à notre guise modifier les paramètres.

```cpp
#include <Eigen/Core>

template<int d>
class PriceIndex
{
    private :
    double T; // Maturité
    double r; // Taux d'intérêt
    Eigen::Matrix<double, d, 1>  s0; // Prix initiaux
    Eigen::Matrix<double, d, 1>  sigma; // Volatilités
    Eigen::Matrix<double, d, d> correlationMatrix; // Matrice de corrélation
    Eigen::Matrix<double, d, 1>  weights; // Poids a_i

    public :
    PriceIndex(double T, double r, Eigen::Matrix<double, d, 1> s0, Eigen::Matrix<double, d, 1>  sigma, Eigen::Matrix<double, d, d> correlationMatrix, Eigen::Matrix<double, d, 1>  weights) : T(T), r(r), s0(s0), sigma(sigma), correlationMatrix(correlationMatrix), weights(weights) {}
};
```

Afin de construire le champ privé `correlationMatrix`, au sein du même fichier `.hpp` se situe un modèle de méthode :

```cpp
template<int d>
Eigen::Matrix<double, d, d> generateCorrelationMatrix(double rho)
```

Nous allons également ajouter un modèle de méthode permettant de simuler des trajectoires de $(S_T^1, \dots, S_T^d)$.

On implémente la méthode `simulateAssetPrices(Z)` où `Z` est un vecteur (préalablement simulé) de d gausiennes centrées réduites indépendantes.

```
{
    public :
    ...
    Eigen::Matrix<double, d, 1>
     ↪ simulateAssetPrices(Eigen::Matrix<double, d, 1> Z)
     ↪ const;
};
```

Comme les coefficients de corrélation ρ_{ij} sont non nuls, il va falloir utiliser une décomposition de Cholesky pour construire $(W_t)_{t\in[0,T]}$, le mouvement brownien dans $\mathbb{R}^d$.

Décomposition de Cholesky : Soit Σ une matrice définie positive (ce qui est le cas pour une matrice de corrélation). Alors il existe une matrice T triangulaire inférieure telle que $\Sigma = TT^t$. Cette décomposition est unique pourvu que les termes diagonaux soient positifs. Alors en prenant $Z \sim \mathcal{N}(0, I_d)$ un vecteur gaussien alors TZ est un vecteur gaussien de $\mathcal{N}(0, TI_dT^t) = \mathcal{N}(0, TT^t) = \mathcal{N}(0, \Sigma)$.

D'autre part, les coefficients de l'EDS (1) sont constants. Dans ce cas on a existence et unicité trajectorielle. Par la formule d'Itô on peut montrer que :

$$S_t^i = S_0^i \exp\left(\left(r - \frac{\sigma_i^2}{2}\right)t + \sigma W_t^i\right), \quad 1 \leq i \leq d.$$

```cpp
#include <Eigen/Cholesky>
#include <random>
#include <cmath>

template<int d>
Eigen::Matrix<double, d, 1>
 ↪  PriceIndex<d>::simulateAssetPrices(Eigen::Matrix<double,
 ↪  d, 1> Z) const
{
    Eigen::Matrix<double, d, 1> S_T;
    Eigen::Matrix<double, d, d> L =
     ↪  correlationMatrix.llt().matrixL();

    Eigen::Matrix<double, d, 1> W_T = L * Z * sqrt(T);
    for (int i = 0; i < d; i++)
    {
        S_T(i) = s0(i) * std::exp((r - 0.5 * sigma(i) *
         ↪  sigma(i)) * T + sigma(i) * W_T(i));
    }
    return S_T;
}
```

3 Méthode de Monte Carlo

On choisit les paramètres suivants :

$$d = 10, \quad T = 1, \quad r = 0.1, \quad s_0^i = 100, \quad \sigma_i = 0.3, \quad a_i = \frac{1}{d}, \quad \rho_{ij} = 0.5, \quad i \neq j$$

Quelque chose d'important à prendre en compte dans la méthode de Monte-Carlo (hormis le temps d'éxécution) est la variance de notre estimateur.

On définit la fonction `MonteCarlo_CallBasket`, déclarée `friend` pour

pouvoir accéder aux champs privés de la classe `PriceIndex`.

```
#include <utility>
{
    public :
    ...
    template<int d_, typename RNG>
    friend std::pair<double, double>
    ↪ MonteCarlo_CallBasket(PriceIndex<d_>& P, double K,
    ↪ RNG& G, int N);
};
```

Elle renvoie une `std::pair<double,double>` (dans `<utility>`), correspondant au couple (valeur estimée, variance). Afin de pouvoir mesurer la variance, implémentons une classe `VarianceCalculator`.

3.1 Algorithme de Welford

L'estimateur sans biais de la variance s'écrit :

$$s_n^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x}_n)^2$$

avec $\overline{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i$, la moyenne de l'échantillon des n premières données. Lorsqu'une donnée est ajoutée, le procédé de mise à jour est le suivant :

$$\overline{x}_n = \frac{(n-1)\overline{x}_{n-1} + x_n}{n} = \overline{x}_{n-1} + \frac{x_n - \overline{x}_{n-1}}{n}$$
$$s_n^2 = \frac{n-2}{n-1}s_{n-1}^2 + \frac{(x_n - \overline{x}_{n-1})^2}{n} = s_{n-1}^2 + \frac{(x_n - \overline{x}_{n-1})^2}{n} - \frac{s_{n-1}^2}{n-1}$$

Ces formules sont instables dans le sens où on fait des soustractions de manière répétée d'un petit nombre et d'un grand nombre qui grandit avec n. Une meilleure quantité pour mettre à jour la somme des carrés des différences

à la moyenne sera :

$$M_{2,n} = M_{2,n-1} + (x_n - \overline{x}_{n-1})(x_n - \overline{x}_n),$$

auquel cas $s_n^2 = \dfrac{M_{2,n}}{n-1}$.

On développe la classe au sein du fichier `variance_calculator.hpp` :

```cpp
class VarianceCalculator {
    private :
    int count;  // Nombre courant de données
    double mean;  // Moyenne courante
    double partial_sum;

    public:
    VarianceCalculator() : count(0), mean(0.0),
    ↪  partial_sum(0.0) {}

    void update(double x) {
        count++;
        double delta = x - mean;
        mean += delta / count;
        double delta2 = x - mean;
        partial_sum += delta * delta2;
    }

    double variance() const {
        return (count > 1) ? partial_sum / (count - 1) : 0.0;
    }

    double meanValue() const {return mean;}
};
```

3.2 Méthode classique

Le corps de la fonction `MonteCarlo_CallBasket` est le suivant :

```
#include "variance_calculator.hpp"
...
template<int d, typename RNG>
std::pair<double, double> MonteCarlo_CallBasket(PriceIndex<d>&
 ↪  P, double K, RNG& G, int N)
{
    double discount_factor = std::exp(-P.r*P.T);
    VarianceCalulator V;
    double sum = 0.0;
    std::normal_distribution<double> normal(0.0,1.0);
    for(int i = 0; i < N; i++)
    {
        Eigen::Matrix<double, d, 1> Z;
        for (int j = 0; j < d; j++) Z(j) = normal(G);
        Eigen::Matrix<double, d, 1> S_T =
         ↪  P.simulateAssetPrices(Z);
        double I_T = P.weights.dot(S_T);
        double x = std::max(I_T - K, 0.0);
        sum += x;
        V.update(x);
    }
    return std::make_pair((sum / N) * discount_factor,
     ↪  V.variance());
}
```

Mesurons les temps avec la bibliothèque `<chrono>` de `C++11`. En prenant $K \in \{80, 90, 100, 110, 120\}$ et $N = 100000$, et en compilant avec l'option `-O2` de `g++` on obtient les résultats suivants :

K	Prix	Variance
80	28.2664	571.462
90	20.5001	492.966
100	14.0598	391.479
110	8.98727	268.987
120	5.63596	178.914

TABLE 1 – Valeur de l'estimateur Monte-Carlo classique, et sa variance. Temps d'éxécution : 0.298813s.

On constate que la variance est très élevée. C'est la raison pour laquelle nous sommes motivés à mettre en oeuvre des méthodes de réduction de variance.

4 Réduction de Variance

4.1 Méthode des variables antithétiques

Principe. Soit X et X' deux variables aléatoires dans $L^1(\Omega, \mathcal{F}, \mathbb{P})$, où on suppose que $X \overset{loi}{=} X'$, avec $\mathrm{Var}(X - X') > 0$. Autrement dit elles ne sont pas indépendantes. On pose

$$\chi = \frac{X + X'}{2}$$

On note respectivement κ_χ et κ_X la complexité de calcul de χ et de X. Ainsi il est naturel que $\kappa_\chi \approx 2\kappa_X$.
On préfère χ à X si $\kappa_\chi \mathrm{Var}(\chi) < \kappa_X \mathrm{Var}(X)$. Ceci équivaut à

$$2\kappa_X \frac{1}{4} \left(\mathrm{Var}(X) + \mathrm{Var}(X') + 2\mathrm{cov}(X, X')\right) < \kappa_X \mathrm{Var}(X)$$

$$\Leftrightarrow \mathrm{cov}(X, X') < 0$$

Proposition. *Soit $Z, \varphi(Z) \in L^1(\mathbb{P})$, avec φ monotone. Soit $T : \mathbb{R} \to \mathbb{R}$ une fonction décroissante telle que $T(Z) \overset{loi}{=} Z$. Alors* $\mathrm{Cov}(\varphi(Z), \varphi(T(Z)) \leq 0$.

Preuve. Il suffit de remarquer que φ et $\varphi \circ T$ sont de monotonie opposée (et

conclure par co-monotonie).

Dans notre cas, on dispose d'un vecteur $(W_T^1, \dots, W_T^d)$. En utilisant la transformation $T(x) = -x$ (ce qui est pertinent d'après la proposition précédente), on utilise la méthode des variables antithétiques avec $(-W_T^1, \dots, -W_T^d)$.

On rajoute au sein du fichier `price_index.hpp` une nouvelle fonction `friend`, tenant compte de ce raffinement :

```
#include <utility>
{
    public :
    ...
    template<int d_, typename RNG>
    friend std::pair<double, double>
    ↪ MonteCarlo_CallBasket_Antithetic(PriceIndex<d_>& P,
    ↪ double K, RNG& G, int N);
};
```

Le code est similaire à celui de `MonteCarlo_CallBasket`. On trouve à présent les résultats suivants :

K	Prix	Variance
80	28.2069	33.3474
90	20.4954	56.1429
100	14.0273	79.0002
110	9.08513	87.2993
120	5.59445	70.2735

TABLE 2 – Valeur de l'estimateur Monte-Carlo avec variables antithétiques, et sa variance. Temps d'éxécution : 0.517279s.

4.2 Méthode des variables de contrôle

Principe. L'idée est de profiter de la corrélation entre la variable d'intérêt X et une (ou plusieurs) variable de contrôle Y, qui est facilement simulable et dont l'espérance est connue. L'estimateur devient alors

$$\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n}\left(X_i - c(Y_i - \mathbb{E}[Y])\right)$$

où (X_i, Y_i) sont les simulations i.d.d. de (X, Y), et c un coefficient de contrôle choisi de manière optimale. On peut montrer que

$$c^* = \frac{\mathrm{cov}(X, Y)}{\mathrm{Var}(Y)}$$

est la constante optimale, pour simplifier ici on prendra $c = 1$.

Dans notre cas, l'option sur la moyenne géométrique est bien corrélée avec une option basket. Le payoff de cette dernière s'écrit :

$$Y = \left(\sqrt[d]{\prod_{i=1}^{d} S_T^i} - K\right)^+$$

On va voir que $\mathbb{E}[e^{-rT}Y]$ admet une formule fermée. Pour cela, on pose $G_T = \sqrt[d]{\prod_{i=1}^{d} S_T^i} = \exp\left(\frac{1}{d}\sum_{i=1}^{d}\ln(S_T^i)\right)$. Déterminons sa loi.

En vertu de l'EDS (1) on sait que

$$\ln(S_T^i) = \ln(S_0^i) + \left(r - \frac{\sigma_i^2}{2}\right)T + \sigma_i W_T^i$$

En sommant on obtient que

$$\frac{1}{d}\sum_{i=1}^{d}\ln(S_T^i) = \frac{1}{d}\sum_{i=1}^{d}\left(\ln(S_0^i) + \left(r - \frac{\sigma_i^2}{2}\right)T\right) + \frac{1}{d}\sum_{i=1}^{d}\sigma_i W_T^i$$

En remarquant que

$$\mathrm{d}\left\langle \frac{1}{d}\sum_{i=1}^{d}\sigma_i W_t^i \right\rangle = \frac{1}{d^2}\sum_{i=1}^{d}\sum_{j=1}^{d}\sigma_i\sigma_j\rho_{ij}\mathrm{d}t$$

Ainsi par le théorème de Lévy, en posant $\sigma^2 = \frac{1}{d^2}\sum_{i=1}^{d}\sum_{j=1}^{d}\sigma_i\sigma_j\rho_{ij}$, le processus

$$B_t = \frac{\frac{1}{d}\sum_{i=1}^{d}\sigma_i W_t^i}{\sigma}$$

est un mouvement brownien standard (en dimension 1). On en déduit alors que

$$\frac{1}{d}\sum_{i=1}^{d}\ln(S_T^i) \sim \mathcal{N}(\mu, \sigma^2 T),$$

avec $\mu = \frac{1}{d}\sum_{i=1}^{d}\left(\ln(S_0^i) + \left(r - \frac{\sigma_i^2}{2}\right)T\right)$. Ainsi G_T suit une distribution log-normale.

Proposition. **(⋆)** *Soient γ, $K > 0$, $\mu \in \mathbb{R}$ et $N \sim \mathcal{N}(0,1)$. Alors*

$$\mathbb{E}\left[(\exp(\mu + \gamma N) - K)^+\right] = e^{\mu + \frac{\gamma^2}{2}}\mathcal{N}(d) - K\mathcal{N}(d - \gamma),$$

avec $d = \dfrac{\gamma^2 + \mu - \ln(K)}{\gamma}$.

Preuve. Voir Appendice.

Notons que contrairement à `scipy.norm.cdf()` de `python`, en `C++` il n'y a pas de fonction pré fabriquée pour évaluer la fonction de répartition d'une gaussienne centrée réduite. On ajoute alors `double norm_cdf(double x)`, utilisant `std::erf` du module `<cmath>`.

Nous sommes maintenant en mesure de déterminer la valeur de $\mathbb{E}[e^{-rT}Y]$, on prend ici $\gamma = \sigma\sqrt{T}$. On implémente la fonction `closed_formula` :

```
{
    public :
    ...
    template<int d_>
    friend double closed_formula(PriceIndex<d_>& P, double K);
};
```

Dont le code se situe une fois de plus dans le fichier `price_index.hpp`.

À l'aide de cette fonction on peut améliorer nos estimations à l'aide de `MonteCarlo_CallBasket_ControlVar` :

```
{
    public :
    ...
    template<int d_, typename RNG>
    friend std::pair<double, double>
     ↪  MonteCarlo_CallBasket_ControlVar(PriceIndex<d_>& P,
     ↪  double K, RNG& G, int N);
};
```

Avec une telle fonction, toujours avec les paramètres on obtient :

K	Prix	Variance
80	28.2304	1.76845
90	20.4881	2.14939
100	14.0314	2.4697
110	9.07893	2.50804
120	5.57761	2.24552

TABLE 3 – Valeur de l'estimateur Monte-Carlo avec variable de contrôle, et sa variance. Temps d'éxécution : 0.31915s.

Le résultat est satisfaisant, néanmoins un mélange entre les deux méthodes donnerait des estimations meilleures.

4.3 Mélange des deux méthodes

L'idée ici est de réaliser une méthode par variable antithétique au sein de la méthode par variable de contrôle. On implémente alors la fonction `MonteCarlo_CallBasket_ControlVar_Antithetic` dont le code complet se trouve dans le fichier `price_index.hpp` :

```
template<int d, typename RNG>
std::pair<double, double>
↪ MonteCarlo_CallBasket_ControlVar_Antithetic(PriceIndex<d>&
↪ P, double K, RNG& G, int N){
    double sum = 0.0;
    ...
    for(int i = 0; i < d; ++i){
        Eigen::Matrix<double, d, 1> Z;
        ...
        // On génère (W_T)
        Eigen::Matrix<double, d, 1> S_T_pos =
        ↪ P.simulateAssetPrices(Z);
        Eigen::Matrix<double, d, 1> S_T_neg =
        ↪ P.simulateAssetPrices(-Z);
        ...
        // Calcul de payoff_X_pos et payoff_X_neg
        double payoff_X = (payoff_X_pos + payoff_X_neg) / 2.0;
        ...
        // Calcul de payoff_Y_pos et payoff_Y_neg par le même
        ↪ procédé que précédemment
        double payoff_Y = (payoff_Y_pos + payoff_Y_neg) / 2.0;
        ...
        sum += payoff_X - payoff_Y;
    }
```

```
    ...
    return ...;
}
```

Avec ce dernier raffinement on obtient les résultats suivants :

K	Prix	Variance
80	28.2289	1.0523
90	20.4951	0.968652
100	14.0357	0.730297
110	9.0780	0.542845
120	5.57814	0.754182

TABLE 4 – Valeur de l'estimateur Monte-Carlo avec variable de contrôle & variables antithétiques, et sa variance. Temps d'éxécution : 0.541246s.

5 Appendice : Preuve de $(\star)$

Proposition. $(\star)$ *Soient* γ, $K > 0$, $\mu \in \mathbb{R}$ *et* $N \sim \mathcal{N}(0,1)$. *Alors*

$$\mathbb{E}\left[(\exp(\mu + \gamma N) - K)^+\right] = e^{\mu + \frac{\gamma^2}{2}}\mathcal{N}(d) - K\mathcal{N}(d - \gamma),$$

avec $d = \dfrac{\gamma^2 + \mu - \ln(K)}{\gamma}$.

Preuve. En utilisant le fait que $\mathcal{N}(d) = \mathbb{P}(N \geq -d)$:

$$\begin{aligned}
\mathbb{E}\left[(\exp(\mu+\gamma N)-K)^+\right] &= \int_{\mathbb{R}} \left(e^{\mu+\gamma x}-K\right)^+ \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\,dx \\
&= \int_{\frac{\ln(K)-\mu}{\gamma}}^{+\infty} e^{\mu+\gamma x}\frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\,dx - K\mathcal{N}(d-\gamma) \\
&= \int_{\frac{\ln(K)-\mu}{\gamma}}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-\gamma)^2}dx \times e^{\mu+\frac{\gamma^2}{2}} - K\mathcal{N}(d-\gamma) \\
&= \int_{\frac{\ln(K)-\mu}{\gamma}-\gamma}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}}dy \times e^{\mu+\frac{\gamma^2}{2}} - K\mathcal{N}(d-\gamma) \\
&= e^{\mu+\frac{\gamma^2}{2}}\mathcal{N}(d) - K\mathcal{N}(d-\gamma)
\end{aligned}$$

□